

Storing Time Series Metrics



Implementing Multi-Dimensional Aggregate Composites with Counters For Reporting

/*

Joe Stein

<http://www.linkedin.com/in/charmalloc>

@allthingshadoop

@cassandranosql

@allthingscala

@charmalloc

*/

Sample code project up at

<https://github.com/joestein/apophis>

Medialets

What we do

Medialets

- Largest deployment of rich media ads for mobile devices
- Over 300,000,000 devices supported
- 3-4 TB of new data every day
- Thousands of services in production
- Hundreds of Thousands of simultaneous requests per second
- Keeping track of what is and was going on when and where used to be difficult before we started using Cassandra
- What do I do for Medialets?
 - Chief Architect and Head of Server Engineering Development & Operations.

What does the schema look like?

```
CREATE COLUMN FAMILY ByDay  
WITH default_validation_class=CounterColumnType  
AND key_validation_class=UTF8Type AND comparator=UTF8Type;
```

```
CREATE COLUMN FAMILY ByHour  
WITH default_validation_class=CounterColumnType  
AND key_validation_class=UTF8Type AND comparator=UTF8Type;
```

```
CREATE COLUMN FAMILY ByMinute  
WITH default_validation_class=CounterColumnType  
AND key_validation_class=UTF8Type AND comparator=UTF8Type;
```

```
CREATE COLUMN FAMILY BySecond  
WITH default_validation_class=CounterColumnType  
AND key_validation_class=UTF8Type AND comparator=UTF8Type;
```

Column Families hold your rows of data. Each row within each column family will be equal to the time period you are dealing with. So an “event” occurring at 10/20/2011 11:22:41 will become 4 rows

```
BySecond = 20111020112141  
ByMinute= 201110201122  
ByHour= 2011102011  
ByDay=20111020
```

Why multiple column families?

http://www.datastax.com/docs/1.0/configuration/storage_configuration

The following attributes can be declared per column family.

Option	Default Value
column_metadata	n/a (container attribute)
column_type	Standard
comment	n/a
compaction_strategy	SizeTiered
compaction_strategy_options	n/a (container attribute)
comparator	BytesType
compare_subcolumns_with	BytesType
compression_options	n/a (container attribute)
default_validation_class	n/a
gc_grace_seconds	864000 (10 days)
key_validation_class	n/a
key_cache_save_period_in_seconds	n/a
keys_cached	200000
max_compaction_threshold	32
min_compaction_threshold	4
memtable_flush_after_mins	ignored in 1.0 and later releases
memtable_operations_in_millions	ignored in 1.0 and later releases
memtable_throughput_in_mb	ignored in 1.0 and later releases
cf_name	n/a (A user-defined value is required)
read_repair_chance	0.1 (repair 10% of the time)
rows_cached	0 (disabled by default)
row_cache_provider	ConcurrentLinkedHashMapProvider
row_cache_save_period_in_seconds	n/a

Ok now how do we keep track of what?

Lets setup a quick example data set first

- The Animal Logger – fictitious logger of the world around us
 - animal
 - food
 - sound
 - home
- YYYY/MM/DD HH:MM:SS GET /sample?animal=X&food=Y
 - animal=duck&sound=quack&home=pond
 - animal=cat&sound=meow&home=house
 - animal=cat&sound=meow&home=street
 - animal=pigeon&sound=coo&home=street

Now what?

Columns babe, columns make your aggregates work

- Setup your code for columns you want aggregated
 - animal=
 - animal#sound=
 - animal#home=
 - animal#food=
 - animal#food#home=
 - animal#food#sound=
 - animal#sound#home=
 - food#sound=
 - home#food=
 - sound#animal=

Inserting data

Column aggregate concatenated with values

2011/10/29 11:22:43 GET /sample?animal=duck&home=pond&sound=quack

- mutator.insertCounter("20111029112243", "BySecond", HFactory.createCounterColumn("animal#sound#home=duck#quack#pond"), 1))
- mutator.insertCounter("20111029112243", "BySecond", HFactory.createCounterColumn("animal#home=duck#pond"), 1))
- mutator.insertCounter("20111029112243", "BySecond", HFactory.createCounterColumn("animal=duck"), 1))
- mutator.insertCounter("201110291122", "ByMinute", HFactory.createCounterColumn("animal#sound#home=duck#quack#pond"), 1))
- mutator.insertCounter("201110291122", "ByMinute", HFactory.createCounterColumn("animal#home=duck#pond"), 1))
- mutator.insertCounter("201110291122", "ByMinute", HFactory.createCounterColumn("animal=duck"), 1))
- mutator.insertCounter("2011102911", "ByHour", HFactory.createCounterColumn("animal#home=duck#pond"), 1))
- mutator.insertCounter("2011102911", "ByHour", HFactory.createCounterColumn("animal#sound#home=duck#quack#pond"), 1))
- mutator.insertCounter("2011102911", "ByHour", HFactory.createCounterColumn("animal=duck"), 1))
- mutator.insertCounter("20111029", "ByDay", HFactory.createCounterColumn("animal#sound#home=duck#quack#pond"), 1))
- mutator.insertCounter("20111029", "ByDay", HFactory.createCounterColumn("animal#home=duck#pond"), 1))
- mutator.insertCounter("20111029", "ByDay", HFactory.createCounterColumn("animal=duck"), 1))

The implementation, its functional

kind of like “its electric” but without the boogie woogie oogie

```
def r(columnName: String): Unit = {
  aggregateKeys.foreach{tuple:(ColumnFamily, String) => {
    val (columnFamily,row) = tuple
    if (row !=null && row.size > 0)
      rows add (columnFamily -> row has columnName inc) //increment the counter
  }
}
```

```
def ccAnimal(c: (String) => Unit) = {
  c(aggregateColumnNames("Animal") + animal)
}
```

```
//rows we are going to write too
aggregateKeys(KEYSPACE \ "ByDay") = day
aggregateKeys(KEYSPACE \ "ByHour") = hour
aggregateKeys(KEYSPACE \ "ByMinute") = minute
```

```
aggregateColumnNames("Animal") = "animal="
```

```
ccAnimal(r)
```

Retrieving Data

MultigetSliceCounterQuery

- `setColumnFamily("ByDay")`
- `setKeys("20111029")`
- `setRange("animal#sound=", "animal#sound=~", false, 1000)`
- We will get all animals and all of their sounds and counts for that day

- `setRange`
`("sound#animal=purr#", "sound#animal=purr#~", false, 1000)`
- We will get all animals that purr and their count

- What is with the tilde?

Sort for success

Not magic, just Cassandra

Codepage layout

[edit]

UTF-8																
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	NUL 0000 0	SOH 0001 1	STX 0002 2	ETX 0003 3	EOT 0004 4	ENQ 0005 5	ACK 0006 6	BEL 0007 7	BS 0008 8	HT 0009 9	LF 000A 10	VT 000B 11	FF 000C 12	CR 000D 13	SO 000E 14	SI 000F 15
1-	DLI 0010 16	DC1 0011 17	DC2 0012 18	DC3 0013 19	DC4 0014 20	NAK 0015 21	SYN 0016 22	ETB 0017 23	CAN 0018 24	EM 0019 25	SUB 001A 26	ESC 001B 27	FS 001C 28	GS 001D 29	RS 001E 30	US 001F 31
2-	SP 0020 32	! 0021 33	" 0022 34	# 0023 35	\$ 0024 36	% 0025 37	& 0026 38	' 0027 39	(0028 40) 0029 41	* 002A 42	+ 002B 43	, 002C 44	- 002D 45	. 002E 46	/ 002F 47
3-	0 0030 48	1 0031 49	2 0032 50	3 0033 51	4 0034 52	5 0035 53	6 0036 54	7 0037 55	8 0038 56	9 0039 57	: 003A 58	; 003B 59	< 003C 60	= 003D 61	> 003E 62	? 003F 63
4-	@ 0040 64	A 0041 65	B 0042 66	C 0043 67	D 0044 68	E 0045 69	F 0046 70	G 0047 71	H 0048 72	I 0049 73	J 004A 74	K 004B 75	L 004C 76	M 004D 77	N 004E 78	O 004F 79
5-	P 0050 80	Q 0051 81	R 0052 82	S 0053 83	T 0054 84	U 0055 85	V 0056 86	W 0057 87	X 0058 88	Y 0059 89	Z 005A 90	[005B 91	\ 005C 92] 005D 93	^ 005E 94	_ 005F 95
6-	~ 0060 96	a 0061 97	b 0062 98	c 0063 99	d 0064 100	e 0065 101	f 0066 102	g 0067 103	h 0068 104	i 0069 105	j 006A 106	k 006B 107	l 006C 108	m 006D 109	n 006E 110	o 006F 111
7-	p 0070 112	q 0071 113	r 0072 114	s 0073 115	t 0074 116	u 0075 117	v 0076 118	w 0077 119	x 0078 120	y 0079 121	z 007A 122	{ 007B 123	 007C 124	} 007D 125	~ 007E 126	DEL 007F 127
8-	• +00 128	• +01 129	• +02 130	• +03 131	• +04 132	• +05 133	• +06 134	• +07 135	• +08 136	• +09 137	• +0A 138	• +0B 139	• +0C 140	• +0D 141	• +0E 142	• +0F 143

What it looks like in Cassandra

```
val sample1: String = "10/12/2011 11:22:33 GET /sample?animal=duck&sound=quack&home=pond"  
val sample4: String = "10/12/2011 11:22:33 GET /sample?animal=cat&sound=purr&home=house"  
val sample5: String = "10/12/2011 11:22:33 GET /sample?animal=lion&sound=purr&home=zoo"  
val sample6: String = "10/12/2011 11:22:33 GET /sample?animal=dog&sound=woof&home=street"
```

```
[default@FixtureTestApophis] get ByDay[20111012];  
=> (counter=animal#sound#home=cat#purr#house, value=70)  
=> (counter=animal#sound#home=dog#woof#street, value=20)  
=> (counter=animal#sound#home=duck#quack#pond, value=98)  
=> (counter=animal#sound#home=lion#purr#zoo, value=70)  
=> (counter=animal#sound=cat#purr, value=70)  
=> (counter=animal#sound=dog#woof, value=20)  
=> (counter=animal#sound=duck#quack, value=98)  
=> (counter=animal#sound=lion#purr, value=70)  
=> (counter=animal=cat, value=70)  
=> (counter=animal=dog, value=20)  
=> (counter=animal=duck, value=98)  
=> (counter=animal=lion, value=70)  
=> (counter=sound#animal=purr#cat, value=42)  
=> (counter=sound#animal=purr#lion, value=42)  
=> (counter=sound#animal=quack#duck, value=43)  
=> (counter=sound#animal=woof#dog, value=20)  
=> (counter=total=, value=258)
```

<https://github.com/joestein/apophis>

A few more things about retrieving data

- You need to start backwards from here.
- If you want to-do things adhoc then map/reduce is better
- Sometimes more rows is better allowing more nodes to-work
 - If you need to look at 100,000 metrics it is better to pull this out of 100 rows than out of 1
 - Don't be afraid to make CF and composite keys out of Time+ Aggregate data
 - 20111023#animal=duck
 - This could be the row that holds ALL of the animal duck information for that day, if you want to look at 100 animals at once with 1000 metrics for each per time period, this is the way to go

Q & A



Medialets
The rich media ad
platform for mobile.



connect@medialets.com
www.medialets.com/showcase